# Bachelor of Software Engineering - Game Programming

# GD2P02 – Physics Programming

## Introduction to Box2D

# Overview

- Box2D
  - Introduction
  - Simple properties
  - Getting Started

# Box2D

- 2D Physics Simulation Engine
- Developed by Erin Catto.
- Free and Open Source
  - Zlib license (permissive free software license)
- Platform Independent

# Box2D used in

- – Crayon Physics (Petri Purho, 2009)
- – Limbo (Playdead, 2010)
- – Rolando (HandCircus, 2008)
- – Fantastic Contraption (Colin Northway, 2008)
- – Incredibots (Grubby Games, 2008)
- – Angry Birds (Rovio Entertainment, 2009)
- – Tiny Wings (Andreas Illiger, 2011)
- – Transformice (Atelier 801, 2010)
- – Happy Wheels (Jim Bonacci, 2010)

# Box2D Features

- Constrained rigid body simulation
  - Convex polygons and circles
  - Multiple shapes
  - Continuous collision detection
  - Pair management
  - Contact manifolds
- Engine supports:
  - Stable stacking with linear time solver
  - Contact, friction, restitution
  - Forces, impulses, momentum

# Box2D Features continued…

- Collision Detection and Collision Resolution
  - Sweep and prune broad phase
    - Sort to limit number of collisions that need checking…
  - Continuous collision detection unit
  - Stable linear-time contact solver

- Comprehensive documentation and forums.
  - http://box2d.org

# Box2D Modules

- Common
  - Allocation, maths, settings…
  - Box2D manages memory allocations…
    - Utilise its factory methods!
- Collision
  - Defines shapes, broad-phase, collision functions and queries…
- Dynamics
  - The simulation of the world, bodies, fixtures, and joints…

# Box2D Units

- Tuned to work with metre-kilogram-second (MKS).
  - Tuned for objects moving between 0.1 and 10 meters.
- Scaling needed for rendering…
- Do not use pixels as your unit…
  - Use metres!
- Radians for angles.

# Box2D Software Development Kit

- Download from http://box2d.org/
- Box2d v2.3.0
  - Extract the zip…
  - The folder Box2d contains the header Box2d.h
  - Place this folder in your project path.
  - **`#include <Box2D/Box2D.h>`**
- Do experiments within the Testbed.
  - Lots of samples…

# Box2D: Shapes

- A 2D geometric object:
  - Such as a circle or polygon.
- `b2Shape` class
  - Test a point for overlap with the shape
  - Perform a ray cast against the shape
  - Compute the shape's AABB
  - Compute the mass properties of the shape

# Box2D: Shapes continued…

- Circle Shapes
  - Solid
  - Have:
    - Position and Radius
  - **`b2CircleShape`** class.

- Polygon Shapes
  - Solid convex polygons.
  - **`b2PolygonShape`** class.
    - Set function takes in vertices…
    - Or **`SetAsBox(…)`**

# Box2D: Shapes continued…

- Edge Shapes
  - – Line Segments
  - – Collides with: Circles and Polygons
    - But not edge shapes!
      - – At least one of the two colliding shapes must have volume…
  - – **`b2EdgeShape`** class

GD2P02

# Box2D: Shapes continued…

- Chain Shapes
  - Efficient way to connect many edges together!
  - Construct static game worlds!
    - Scrolling game world…
  - Provides two sided collision.
  - **`b2ChainShape`** class
  - Automatic loop creation:
    - **`chain.CreateLoop`**`(…)`
  - Self-intersection of chain shapes not supported!

# Box2D: Shape Point Test

- Test a point for overlap with a shape
  - Need:
    - Transform for the shape…
    - Point in world space…
  - For example:

```
b2Transform transform;
transform.SetIdentity();
b2Vec2 point(3.0f, 4.0f);
bool bHit = shape->TestPoint(transform, point);
```

# Box2D: Ray Cast

- Cast a ray at a shape…
  - Find the first point of intersection, and the normal vector.
  - No hit if the ray starts inside the shape!

```
b2RayCastInput
b2RayCastOutput
bool bHit = shape->RayCast(&output,
input, transform, childindex);
 childindex = 0;
```

# Box2D: Testing for Overlap

```
b2TestOverlap(shapeA, indexA,
shapeB, indexB, transformA,
transformB);
```

– Provide child indices for the chain shapes.

# Box2D: Contact Manifold

- Compute contact points for overlapping shapes.
- If contact points share the same normal, then they are grouped into a manifold.
  - `b2Manifold` class
    - Holds: Normal, two contact points.
    - Local coordinates.

# Box2D: Fixtures

- A fixture binds a shape to a body.
- Properties can be added:
  - Density
  - Friction
  - Restitution
  - Parented to a body…
  - **`b2FixtureDef`** class.

# Box2D: Constraints

- A physical connection that removes degrees of freedom from bodies.
- 2D bodies have 3 degrees of freedom:
  - Translate x.
  - Translate y.
  - Rotate.
- Contact Constraint
  - Prevents penetration of rigid bodies.
  - Simulate friction and restitution.

# Box2D: Worlds

- A physics world is a collection of bodies, fixtures, and constraints that interact together.
- Factory:
  - `CreateBody(…)`
  - `CreateJoint(…)`
  - `DestroyBody(…)`
  - `DestroyJoint(…)`
- Also:
  - `GetBodyList()`
  - `ClearForces()`

GD2P02

# Box2D: Creating a World

- Define a gravity vector:
  - `b2Vec2 gravity(0.0f, -10.0f);`
- Create the world:
  - `b2World myWorld(gravity, true);`
  - Second parameter enables sleeping objects.
    - Allows bodies to sleep when they come to rest.
      - Hence no updating…
- Or:

```
m_pWorld = new b2World(gravity, bSleep);
…
delete m_pWorld;
```

# Box2D: Creating “Ground”

- Create a body!
  - Define a body
    - Position, damping, etc.
  - Use the world object to create the body.
  - Define fixtures with a shape, friction, density, etc…
  - Create fixtures on the body.

  - Bodies are static by default…
    - Which is perfect for the ground…

# Box2D: Creating Dynamic Bodies

- Similar to creating the ground…
- But…
  - Enable dynamic!
- `b2BodyType`
  - `b2_dynamicBody`
- Now the body will move in response to forces!

# Box2D: Bodies

- Mass: How heavy the body is.
- Velocity: How fast and the direction of movement.
- Rotational Inertia: How much effort to start or stop spinning.
- Angular Velocity: How fast and which way the body is rotating.
- Location: Where the body is.
- Angle: Which way the body is facing.

- **`b2BodyType: b2_dynamicBody`**

# Box2D: Bodies

- Rigid Body
  - A chunk of matter.
  - Strong enough that the distance between any two bits of matter on the chunk is always constant.

- Body
  - Factory Method:
    - `CreateFixture(…)`
    - `DestroyFixture(…)`

# Box2D: Creating Bodies

- Set up a definition.
- Create the body from the definition.
  - Make multiple bodies from a single definition…
- For example:

```
b2BodyDef bd;
bd.type = b2_dynamicBody;
bd.position.Set(0, 30);

b2Body* pDynamicBody =
            m_pWorld->CreateBody(&bd);
```

# Box2D: Joints

- Constraint used to hold two or more bodies together.
- Box2D supports:
  - Revolute Joint: Hinge, pin, or axle…
    - An anchor point is defined on each body…
    - The bodies will move so that these two points are always in the same place.
  - Prismatic Joint (Slider joint): Elevator, moving platform, piston…
    - Two bodies have their rotations held fixed relative to each other.
    - They can only move along a specific axis…
  - Distance Joint: distance constraint...
  - Gear Joint, Wheel Joint etc.

# Box2D: Joints

- Joint Limit
  - Restricts the range of motion of a joint.
- Joint Motor
  - Drives the motion of the connected bodies.

# Box2D: Time Step

- Generally at least 1/60 seconds (60Hz).
- Avoid variable time steps!
  - Keep it constant.
    - Easier to debug!!!
- Avoid coupling the time step to the frame rate!

```
m_pWorld->Step(fTimeStep, velIter, posIter);
```

# Box2D: Debug Draw

- Derive a class from **`b2Draw`**.
- Implement the methods:
  - **`DrawPolygon`**
  - **`DrawSolidPolygon`**
  - **`DrawCircle`**
  - **`DrawSolidCircle`**
  - **`DrawSegment`**
  - **`DrawTransform`**
- Box2D will call your implementations to use your renderer to visualise the physics simulation.
  - Very useful for debugging!

# Box2D: Logging

- Logging
  - Function: **b2Log**
    - Inside: b2Settings.cpp

```
void b2Log(const char* string, ...)
{
    va_list args;
    va_start(args, string);
    vsprintf_s(string, args);
    va_end(args);
}
```

  - Modify this function to utilise your logging technique…

# Box2D: Getting Started

- Include <Box2d/Box2d.h>
- Check the project compiles and links…
- Create a world (**`b2World`**)
- Create your debug draw object and activate it…
  - Call **`SetFlags(b2Draw::e_shapeBit);`**
- Attach your debug draw object to your world instance:
  - **`m_pWorld->SetDebugDraw(myDebugDraw);`**

# Box2D: Getting Started continued…

- Create `b2BodyDef`, `b2PolygonShape`, `b2FixtureDef`
  - Ultimately call `m_pWorld->CreateBody(…);`
- To update the simulation:
  - Call `m_pWorld->Step(…);`
- To draw the debug data:
  - Call `m_pWorld->DrawDebugData();`
- Remember to check for memory leaks…
  - Clean up as necessary…

GD2P02

# Box2D: Documentation

- Well documented
  - Well commented.
  - http://www.box2d.org/documentation.html
  - Doxygen API Documentation

- Good Community
  - Check out the forums…
  - http://www.iforce2d.net/b2dtut/testbed-structure

## Summary

- Box2D
  - Introduction
  - Simple properties
  - Getting Started